CARNEGIE-MELLON UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

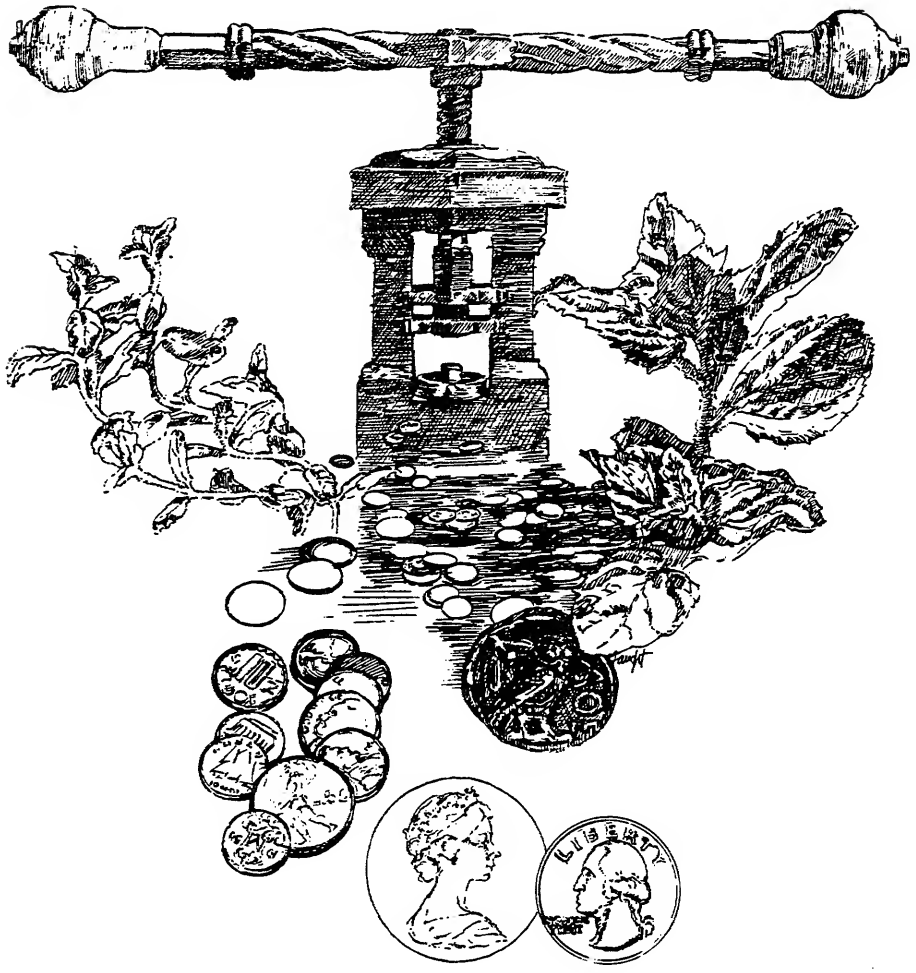
SPICE PROJECT

User Manual for Mint —

The Spice Document Preparation System

Penny Anderson, Peter Hibbard, and Kathryn Porsche

14 February 84



14 February 1984
Location of machine-readable file: prman.mss

Supported by the Defense Advanced Research Projects Agency, Department of Defense, ARPA Order 3597, monitored by the Air Force Avionics Laboratory under contract F33615-81-K-1539. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Projects Agency or the U.S. Government.

Table of Contents

Part One
Getting Started

This document describes version 2B(12) of Mint, a document preparation system that has been written as part of the Spice project. Mint has been written as a research vehicle for exploring document preparation, and interactive document preparation in particular. Although the current version of Mint does not have interactive features, it is nonetheless a usable tool which is suitable for release to a wider community for use and evaluation. In making this release, I am making a commitment to providing a stable and maintained system.

The document is organized as follows. This introduction provides an overview of the system and gives operating information; it is followed by a brief review of Mint for Scribe users. The information provided should be sufficient to allow the casual user to prepare documents on the Perq of the same quality as those produced by Scribe. More detailed information can be found in the Reference Manual.

1.1 Mint and Scribe

At a superficial level, Mint resembles Scribe[1] — it takes as input a `.Mss` file, and produces a formatted document for some device. Most `.Mss` files that just use the basic Scribe commands — those in sections 1 to 6 of the Scribe manual — will be accepted by Mint, and will produce results that resemble those from Scribe. In addition, there are Mint equivalents for most of the rest of the Scribe facilities, though these are obtained in different ways.

1.2 Getting Started

I assume that you already have access to a Perq, and are sufficiently familiar with the machine to boot it, run programs, create and edit files, etc. Other parts of the User Manual describe how to do these.

---

1 Scribe is a registered trademark of UNILOGIC, Ltd.

1.2.1 Loading Mint onto a Perq

Normally Mint will be on the Perq, and there will be no need to reload it. If, however, Mint has not yet been installed, or the version currently available on the Vax is a more recent version than that on the Perq, then you should load it as follows:

```
@update mintrun<RETURN>
```

(when complete, Update will display the Mint change log). If Update cannot be found the Perq will return an error message:

```
** Link-F-Update.Run not found
```

If that happens, you must retrieve a new version of Update. The section of this User Manual that describes Update tells you how to do this.

1.2.2 Running Mint

To invoke Mint, type

```
mint
```

If the Perq responds with

```
** Loader-F-Mint.RUN not found
```

then Mint has not been loaded, and you should start again at section 1.2.1. If Mint is loaded, it will request you to create a new window. This window will be used to display the formatted output, though your dialogue and any error messages will appear in the first window.

Mint prompts for the name of the file to be formatted by asking

```
File to be read from:
```

The file name can be typed in as a full path name or just as the file itself, in which case Mint will search for the file on the current search list. The extensions `.Mss` and `.Mint` may be omitted.

After it has asked for the file, Mint will ask for which device the output is intended.

```
Output to Perq, Dover, other (P, D, X, <string>) [X]
```

Currently there are two devices available — the Perq itself (type P), and the Dover (type D). Mint then asks a couple of questions about debug output — respond with carriage return to both questions.

```
Debug output onto screen (Y or N) [N]:
Debug output into file (Y or N) [N]:
```

Mint then goes about its business. As it formats the document it places the current file and line number in the banner of the original window, but as it only does this on each change of environment, there may be periods of several seconds when the screen does not change. If the output device has been specified to be the Perq, the document galleys are written into the new window. Section numbers will appear as question marks, and several galleys may get overlaid. Do not worry about this — what you are seeing is only an intermediate version of the document. Error messages appear in the original window and are written off to an error file. More details are given in the next section.

When the document has been completely formatted, Mint will ask the question

```
Print on Perq, Dover, other; Report, Quit (P, D, <string>; R,Q) []:
```

You now have the option of choosing the device on which to view the document. If you specify the Dover, then a press file will be created, with extension .Press, which can be shipped to the Dover for printing. If you specify the Perq, the messages

```
Which part: TitlePage, Contents, Mainbody (T, C, M) [M]:
Page: number, All, Dover, Quit, next (<integer>, A, D, Q, <CR>) []:
```

will appear. The first prompt asks which part of the document you want to see. Normally you will want the main body, so just type a carriage return. Typing in a page number to the second prompt will cause that page to be displayed on the screen, typing A or a will cause all the pages to be displayed, and typing a carriage return will cause the next page to be displayed.

Text cross-proofed on the screen may look somewhat odd since Mint positions each character exactly where it would appear on the printed page. The screen is smaller than the actual printed page so characters may be closer together than they actually will be on paper.

You may exit by typing Q or q, when the question

```
Print on Perq, Dover, other; Report, Quit (P, D, <string>; R,Q) []:
```

reappears. If you exit without creating a press file, you will have to run Mint all over again to produce a press file for printing on the Dover. If you type R or r then a bug report will be prepared for the maintainer (i.e., Peter Hibbard); more details are given in section 1.3.

Note that the viewing device may differ from the target device given in the first request. This allows you

to "cross-proof" a document intended for one device on another. Normally you would want to view Dover output on the Perq screen in order to save the delay in shipping a Press file to the Dover; if it is satisfactory you can create the Press file without reformatting the document. If you are cross-proofing on the Perq, you can also select which page you want to incorporate into the Press file by typing the characters `D` or `d` when Mint asks for which page to print.

In cross-proofing mode, each character appears on the viewing device in the position that it will have on the target device, and diagrams are scaled appropriately. However, the fonts used on the two devices will not be identical, so that the output will not be an exact representation. The Reference Manual explains how to map different fonts to improve the appearance of cross-proofed documents.

1.3 What to do in case of trouble

Error messages generated by Mint fall into four classes.

`Warning`	Mint issues a warning if it finds something suspicious in the input, but which is quite legal. The output from Mint may be satisfactory, but you should investigate the reason for the warning, as it may indicate some misunderstanding.
`Error`	Errors occur when Mint is not able to process the text as you ask, but it is able to take some corrective action and continue. It is unlikely that the output will be what is desired.
`Heresy`	A heresy indicates that there is some serious problem that Mint is not able to fix in a reasonable way. In general these are caused by internal problems, and usually indicate Mint bugs, though improper use of the advanced facilities described in the Reference Manual also can cause them. As in all organizations, one can continue after a heresy, though subsequent actions by the system cannot be predicted.
`Fatal Error`	These occur when Mint has discovered an internal error that will cause it to fail if it continues. Usually these are caused by overflow of internal tables, and can be fixed fairly easily in future releases.

In all cases, the error message is written in the original window, and sent off to a file with the extension `.Error`, with an indication of the location of the error. In the case of a warning or an error, Mint continues; in the case of a heresy or fatal error it halts with the message

```
Quit, Continue, Report or Alter Flags (Q, C, R, A) [R]:
```

Unless you are a Mint maintainer you should type `Q` or `q` (or `R` or `r`, which under these circumstances have the same effect).

After a fatal error or a heresy, Mint will prepare a report for the maintainer and use the mail system to deliver it. Mint will ask permission to include the `.mss` file in the report along with the `error` file.

To stop Mint, you can type `↑C` if you are using POS and the message

```
Quit, Continue, Report or Alter Flags (Q, C, R, A) [C]:
```

will appear. Quit, continue or report, as you wish. (You can alter the Debug Flags also, but then you are out on your own). Under Accent, type `↑DEL C` to kill Mint; no message will appear as control is returned to the Shell.

Part Two
A Scribe-oriented Reference Manual

This section provides a brief description of differences between Mint and Scribe, organized along the lines of Appendix E of the Scribe User Manual. It is intended to help Scribe users get started with Mint. The Scribe novice should be able to use Mint with the aid of this reference combined with the Scribe documentation.

2.1 General Guidelines

In general, you should not expect Mint to behave exactly like Scribe under any conditions. The same sorts of things can be done, in roughly the same ways, but the final product will almost never look like what Scribe produces. It should, of course, look better.

Mint's command syntax can be confusing to the Scribe user. Many Scribe commands, such as `@caption`, are environments in Mint, and are used in the same way as other environments, such as `@heading` and `@itemize`. A few constructs, such as `@begin` and `@end` are built-in commands. The rest, such as `@make` and `@include`, are macrogenerator commands. These require that parameter names be either omitted, or separated from their corresponding values by an equals sign. If you have trouble, refer to the Reference Manual.

2.2 Mint Goodies

Some reasons to use Mint instead of Scribe:

Borders	You can draw borders round text to set off figures and tables. See the Reference Manual for examples and an explanation of how to use them.
Page headings	Much more flexible page headings and footings are available in Mint. For an example see the reference manual which shows how the headings and footings in this Reference Manual were obtained.
DP and Plot	The output of DP and Plot can be incorporated directly into a document.
Mathematics	Good quality mathematical typesetting is available via the `@maths` and `@m` environments.

State Files — Modifications which you make tvo Mint's default choices about the style of a document may be saved in a state file, which may then be used to control the style of other documents you develop. See the Reference Manual for details of how to use them.

Cross-proofing — Although Mint is not yet interactive you don't have to print your document to get an idea of what it will look like. Instead, you can simply tell Mint that the document is destined for a Dover, but that you wish to view it on the Perq screen.

Kerning and Ligatures — These are now used where appropriate. The Reference Manual contains details.

2.3 Document Types

To select the document type, use the `@make` command with the syntax:

```
@make(document-type, optional-document-form, optional-device)
```

Warning: don't write `@make(article, form 1, ...)`; this will be misinterpreted. Instead, write `@make(article, form=1, ...)` or `@make(article, 1, ...)`. If specified, `device` refers to the target device (not the viewing device which may be different if you are cross-proofing), and may be overriden when you start up Mint.

Mint supports the Scribe-like document types `Text, form=0` or `form=1; Article, form=0` or `form=1; Report, form=0` or `form=1; Manual, form=0` or `form=1; Thesis`; and `Slides`. As an example, `Manual, form=1` formatted this document.

There is not as yet any support for the Scribe document types `brochure, guide, letter, letterhead`, or `referencecard`.

2.4 Environments

Most Scribe environments are also provided in Mint. Mint also implements some environments not available in Scribe; a complete list is given in the reference manual, section 4.2.3.

Scribe environments provided in Mint, with some differences, are:

B, C, G, etc. — Fonts are selected in Mint pretty much as in Scribe. The default font for an environment may not always be what you expect, however.

`Center`	May also be spelled "centre".
`Description`	This environment will work as expected unless you try to use `@multiple`. If you must have multiple paragraphs within a description item, see the example in section 2.5.
`Example`	Works pretty much like Scribe, but may not give you the font you expect.
`FlushLeft`	Like Scribe.
`FlushRight`	Like Scribe.
`Format`	Like Scribe.
`Heading`	Like Scribe.
`Itemize`	There are two things to watch out for here. First, there are no bullets available on the Perq. Second, nested itemizations and enumerations require the use of `@multiple`. See the example in section 2.5.
`MajorHeading`	Like Scribe.
`Multiple`	Has the same general meaning as in Scribe, but will be required under different cirumstances.
`ProgramExample`	Currently identical to `Example`. This should eventually become a pretty-printer.
`Quotation`	Like Scribe.
`Subheading`	Like Scribe.
`TitlePage`	The Mint titlepage environment has a different syntax from that of Scribe. All title page text must be enclosed in one of the sub-environments, `TitleBox`, `ResearchCredit`, `Abstract`, `CopyrightNotice`, or `Notice`. The Spice title page definition in the file `SpiceTitlePage.Lib` provides a helpful, if somewhat elaborate, example.
`Verbatim`	Like Scribe, except possibly for the choice of font.
`Verse`	Similar to Scribe's verse environment, but results in a different look.

The mathematics-oriented Scribe environments do not occur in Mint, since their capabilities are supplied through other mechanisms. For instance, the `Definition`, `Equation`, `Lemma`, `Proof`, and `Proposition` environments can all be replaced by the use of the appropriate formatting environment (which might be `maths`), together with user-defined counters, which are described in the Reference Manual.

The `Group` environment does not exist in Mint. A similar, but not identical effect can be obtained via:

```
@begin(multiple, need all)
...text to be kept together...
@end(multiple)
```

`Text` in Mint is a document type, *not* an environment. In some cases where you must use the `text` environment in Scribe, no special environment is needed in Mint (e.g., in figures.) In other cases, you might try using the `Default` environment.

Some Mint environments that are not available in Scribe are:

`Align`	Very useful for tables. This environment must be used if you need to center or flush right with respect to tab stops. Two particularly treacherous differences: first, Mint centers text *on* a tab stop where Scribe centers *between* tab stops; second, text to be flushed right to a tab stop should in general be entered as `@w(text)`.
`Commentary`	With `Gloss`, allows you to place two boxes side-by-side, with the right hand side (the gloss) in a smaller font.
`Describe`	A more general way to place environments side by side than `Commentary`.
`DP, Plot`	For including drawings produced by the DP and Plot programs.
`Maths`	For mathematical typesetting comparable to that of TEX.

2.5 Mint Environment Examples

One way to allow for more than one paragraph in a description-like environment (this example corresponds to Figure 3-3 of the *Scribe User's Manual*):

```
@begin(describe)
Segment

@begin(default)
One of the parts into which something naturally separates or is
divided; a division, portion, or section.
@end(default)

Section

@begin(multiple)
A part that is cut off or separated; a distinct part or subdivision
of anything, as an object, country, or class.

The act of subdividing some object into its distinct parts.
@end(multiple)
@end(describe)
```

One way to get nested enumerations; the same principle applies to nested itemizations (this example corresponds to Figure 3-2 or the *Scribe User's Manual*):

```
@begin(enumerate)
@multiple{The numbers in an enumerate environment are filled in by Scribe.
Some styles use roman numerals instead of numbers.
@enumerate[When you nest one Enumerate inside another, the numbers and
margins are adjusted appropriately.

The switch from numbers to letters is specified as part of the document type.
Deeper nesting will produce other kinds of numbering.]}

@begin(multiple)
Normally each blank line starts a new item because each paragraph is an item.

When you need more than one paragraph in an item, use the ...
@end(multiple)

In an enumerated list, Scribe generates the numbers automatically...
@end(enumerate)
```

2.6 Scribe Commands

Some Scribe commands correspond to Mint environments. For example, `caption` is an environment, not a command. This means that `@begin(caption)`, which is unacceptable in Scribe, is accepted by Mint as well as the Scribe form `@caption(...)`.

The Scribe commands which correspond to some Mint construct are:

`Begin`	...is the same in Mint, except that the attribute-value list is somewhat different. The attributes are discussed in the Reference Manual.
`Bibliography`	Mint supports bibliographies as described in the Reference Manual.
`Blankpage`	Provided as a Mint macro.
`Blankspace`	Not defined. Use the macro `@vsp` instead. Other macros for defining blank space are `@hsp` for horizontal space, and `@newline` for blank lines.
`Caption`	is a Mint environment.
`Define`	like Scribe. See the Reference Manual.
`Device`	There is no such Mint command. A device can be specified in the `@make` command or when prompted by Mint.
`Foot`	is a Mint environment.
`Include`	is a Mint macro.
`Index`	The `@index` command has the same meaning as in Scribe, but also has a parameterized form.

`Label`	is a Mint macro similar in function to Scribe's labels. See the Reference Manual for a discussion of Mint's support for labels and counters.
`Modify`	like Scribe.
`NewPage`	is a Mint macro with a similar function to Scribe's.
`PageFooting`	and `PageHeading` are Mint environments. Mint does not number pages automatically If you simply want the page number of the current page to appear in, say, a footing, use `@pagefooting(@pageno())`.
`PageRef`	See the Mint Reference Manual for how to obtain page and other cross-references in the text. A short example: `@nconv(Arabic, PageNo, YourLabel)` obtains the page number, in arabic numerals, associated with `YourLabel`.
`TabClear`	`TabDivide`, and `Tabset` are not commands in Mint; they are box environment parameters. Examples: `@begin(default, tabclear...)` and `@begin(example, tabset 1in, 2in, 3in)`.
`Use`	Generally, Mint uses `@include` for most references to other files, including bibliographies.

Appendix A
Customizing documents

This appendix contains more detailed information than the average Mint user needs to know. The first section is an overview of the facilities that need to be on a Perq in order for Mint to run. The second section explains how to adapt an existing document format to your particular needs. These adaptions can be added to the rest of your library files. A certain amount of familiarity with Mint and with the concept of creating document lay-outs is assumed.

A.1 Mint Files

This section describes two files and three groups of files that should be kept on a Perq if one expects to format a document with Mint. These files are as follows:

- `Mint.Run` (Mint itself)
  This should be the most recent version. The second page of this manual explains how to retrieve an up-to-date version from the CFS-Vax.
- `Fonts.Width`
  This file contains information about the letter widths for all the fonts used by the Dover.
- `State files`
  These files contain high-level specifications about how the different document types should look. These files have the extension `.state`. More will be said about these files later on in this section.
- `Perq font files`
  Files with the extension `.kst` used to produce the text on a Perq screen.
- `Library files`
  Library files are a collection of Mint statements that typically alter existing environments, specify new environments, or define macros; such files have the extension `.lib`. The current library files are listed below.

`Cmr10Kern`	defines information for improving the appearance of documents that use the CMR family of TEX fonts
`ContentsMacros`	contains definitions needed for formatting a table of contents
`CrossProofTR`	defines which Perq fonts will be used when cross-proofing a document that uses the Dover's TimesRoman fonts
`CrossProofSlides`	similar to CrossProofTR except that it is for slides document types

`SpaceFont`	allows modification in the spacing between letters; this library will produce larger or smaller spacing than normal
`SpiceTitlePage`	contains definitions for formatting the title page for any Spice title page.
`StdAlphabeticRefs`	contains definitions for formatting standard alphabetic bibliograhpic references
`StdNumericRefs`	contains definitions for formatting standard numeric bibliographic references
`TimesRomanKern`	defines information for improving the appearance of documents using the TimesRoman family of fonts

A.2 More about State files

This section explains some general concepts about formatting documents and how to customize a document lay-out. The use of state files allows one to produce documents having various formats. These files design a document in accordance with the three variables of document type (text, article, report,etc.), document form (0 or 1 — this denotes slightly different forms of a document type), and output device (Dover or Perq). The properties of a state file can be surmized from its generic name of `TypeFormDevice.state`. One designates the lay-out of a document (or more accurately, one specifies a state file) with the `@make` command at the beginning of the manuscript. The `make` statement takes the three parameters listed above: for example,

```
@make (report,0,dover)
```

This instructs Mint to go off and read the corresponding state file (in this case `report0dover.state`). Mint also provides default parameters; if the second parameter is not specified, the default is `0`. The default for the third parameter is `perq`. If the `make` statement is entirely omitted, Mint will supply a `make` statement with the parameters `text`, `0`, and `perq`. A word of caution: there are no defaults for a parameterless `make` statement. A `make` statement without parameters is used in constructing a new state file; hence, it does not read any existing state file. The form `@make()` is intended for system maintainers only.

Customizing a specific lay-out is possible through the use of the `readdefs` statement. The high-level effect of a `readdefs` command is to add libraries, macros, definition files, etc. to an existing state file. A file containing these modifications is created by the user and given the extension `.defs`. The `readdefs` statement takes the argument of a filename having the extension of `.defs`.

```
@readdefs (your.defs file)
```

The first command in a `defs` file should be the `@make` command–this will specify the state file to be

modified. Actually, the state file itself is not altered; modifications are added to the state file so that the document format defined by the `defs` files is a variation on the `state` file's format. The rest of the file consists of several lines of commands that will specifing the changes to be made to the state file. These commands can be `@library`, `@include`, `@form`, or any `Mint` command. Obviously, a variety of `defs` files can be constructed for a number of tailored formats. For instance, if the user wanted to add several library files, modify commands, etc. to the state file `report0dover.state`, he would create the file called `report.defs` as follows:

```
@make(report,0,dover)
@library(ContentsMacros)
@library(StdNumericRefs)
@include(myreport)
@form(part,p,f,)
@modify(Figure, BorderStyle=LineAboveAndBelow, Border 0.05in)
@modify(Table, BorderStyle=LineAboveAndBelow, Border 0.05in)
@define(code=example,
         fontsize=N, facecode=T, border=0.05in,
         leftmargin=0in, rightmargin=0in)
```

To use the above `.defs` file, the first manuscript command would be

```
@readdefs(report)
```

The document would basically be formatted in accordance with `report0dover.state` excepting those customizations found in `report.defs`.